在优化问题中，有两个关键点

**代价函数：**

确定问题的形式和规模之后，根据不同的问题，选择要优化的目标。

如第一个问题中，优化目标是使得航班选择最优，共计12个航班，要使得总的票价最少且每个人的等待时间之和最小。

第二个问题是学生选择宿舍的问题，每个学生可以实现填报志愿，如果安排的宿舍与志愿完全一致，则代价为0，与第二志愿一致，代价为1，如果没有和志愿一致，代价为3。

故，抽象问题的能力很重要，如何将自己要优化的目标量化的表达出来，是解决优化函数的关键。如在普通的数值优化问题中，可选择当前值与目标值距离的绝对之差，或者使用平方损失函数，均可。

**值域：**

值域是确定搜索的范围。一个可行解的范围是多少，比如学生选宿舍的问题，一共学校有100间宿舍，编号为1-100，那么这个数值优化问题的每个可行解的范围均在100之内。不同的问题对应不同的可行解，也要具体问题具体分析。

**随机搜索算法**

随机搜索算法是最简单的优化搜索算法，它只适用于代价函数在值域范围内没有任何变化规律的情况。

即找不到任何使得代价下降的梯度和极小值点。我们只需要在值域范围内生成足够多的可行解，然后分别计算每个可行解的代价，根据代价选择一个最小的可行解作为随机搜索的最优解即可。

**爬山法**

爬山法假设当前解和周围的解是有变化规律的，如，当前解的下方有一个代价较小的解，则我们就认为，沿着这个方向走，解会越来越小。

步骤为：首先选择一个解作为种子解，每次寻找与这个解相近的解，如果相近的解中有代价更小的解，则把这个解作为种子解。

而如果周围的解都比该解的代价大，则表示已经到达了局部极小值点，搜索停止。

**模拟退火算法**

从一个问题的原始解开始，用一个变量代表温度，这一温度开始时非常高，而后逐步减低。

在每一次迭代期间，算法会随机选中题解中的某个数字，使其发生细微变化，而后计算该解的代价。关键的地方在于计算出该解的代价后，如何决定是否接受该解。

如果新的成本更低，则新的题解就会变成当前题解，这与爬山法类似；如果新的成本更高，则新的题解与概率 P 被接受。这一概率会随着温度T的降低而降低。即算法开始时，可以接受表现较差的解，随着退火过程中温度的不断下降，算法越来越不可以接受较差的解，直到最后，它只会接受更优的解。

其中P = exp[-（newcost - oldcost）/ T ]

其中newcost是新解的成本，oldcost是当前成本，T为当前温度。算法以概率P接受新的解。

|  |
| --- |
| # 搜索算法4：模拟退火算法  # 参数：T代表原始温度，cool代表冷却率，step代表每次选择临近解的变化范围  # 原理：退火算法以一个问题的随机解开始，用一个变量表示温度，这一温度开始时非常高，而后逐步降低  # 在每一次迭代期间，算法会随机选中题解中的某个数字，然后朝某个方向变化。如果新的成本值更低，则新的题解将会变成当前题解，这与爬山法类似。不过，如果成本值更高的话，则新的题解仍有可能成为当前题解，这是避免局部极小值问题的一种尝试。  # 注意：算法总会接受一个更优的解，而且在退火的开始阶段会接受较差的解，随着退火的不断进行，算法越来越不能接受较差的解，直到最后，它只能接受更优的解。  # 算法接受较差解的概率 P = exp[-(highcost-lowcost)/temperature]  def annealingoptimize(self, domain, T=10000.0, cool=0.98, step=1):  # 随机初始化值  vec = [random.randint(domain[i][0], domain[i][1]) for i in range(len(domain))]  # 循环  while T > 0.1:  # 选择一个索引值  i = random.randint(0, len(domain) - 1)  # 选择一个改变索引值的方向  c = random.randint(-step, step) # -1 or 0 or 1  # 构造新的解  vecb = vec[:]  vecb[i] += c  if vecb[i] < domain[i][0]: # 判断越界情况  vecb[i] = domain[i][0]  if vecb[i] > domain[i][1]:  vecb[i] = domain[i][1]  # 计算当前成本和新的成本  cost1 = self.schedulecost(vec)  cost2 = self.schedulecost(vecb)  # 判断新的解是否优于原始解 或者 算法将以一定概率接受较差的解  if cost2 < cost1 or random.random() < math.exp(-(cost2 - cost1) / T):  vec = vecb  T = T \* cool # 温度冷却  print vecb[:], "代价:", self.schedulecost(vecb)  self.printschedule(vec)  print "模拟退火算法得到的最小代价是：", self.schedulecost(vec)  return vec |

**遗传算法**

借鉴生物进化论，遗传算法将要解决的问题模拟成一个生物进化的过程，通过复制、交叉、突变等操作产生下一代的解，并逐步淘汰掉适应度函数值低的解，增加适应度函数值高的解。这样进化N代后就很有可能会进化出适应度函数值很高的个体。

举个例子，使用遗传算法解决“0-1背包问题”的思路：0-1背包的解可以编码为一串0-1字符串（0：不取，1：取） ；首先，随机产生M个0-1字符串，然后评价这些0-1字符串作为0-1背包问题的解的优劣；然后，随机选择一些字符串通过交叉、突变等操作产生下一代的M个字符串，而且较优的解被选中的概率要比较高。这样经过G代的进化后就可能会产生出0-1背包问题的一个“近似最优解”。

**编码**：需要将问题的解编码成字符串的形式才能使用遗传算法。最简单的一种编码方式是二进制编码，即将问题的解编码成二进制位数组的形式。例如，问题的解是整数，那么可以将其编码成二进制位数组的形式。将0-1字符串作为0-1背包问题的解就属于二进制编码。

遗传算法有3个最基本的操作：选择，交叉，变异。

**选择**：选择一些染色体来产生下一代。一种常用的选择策略是 **“比例选择”**，也就是个体被选中的概率与其适应度函数值成正比。假设群体的个体总数是M，那么那么一个体Xi被选中的概率为f(Xi)/( f(X1) + f(X2) + …….. + f(Xn) ) 。比例选择实现算法就是所谓的“轮盘赌算法”( Roulette Wheel Selection ) ，轮盘赌算法的一个简单的实现如下：

|  |
| --- |
| 轮盘赌算法  /\*  \* 按设定的概率，随机选中一个个体  \* P[i]表示第i个个体被选中的概率  \*/  int RWS()  {  m =0;  r =Random(0,1); //r为0至1的随机数  for(i=1;i<=N; i++)  {  /\* 产生的随机数在m~m+P[i]间则认为选中了i  \* 因此i被选中的概率是P[i]  \*/  m = m + P[i];  if(r<=m) return i;  }  } |

**交叉(Crossover)**：2条染色体交换部分基因，来构造下一代的2条新的染色体。例如：

交叉前：

00000|011100000000|10000

11100|000001111110|00101

交叉后：

00000|000001111110|10000

11100|011100000000|00101

染色体交叉是以一定的概率发生的，这个概率记为Pc 。

**变异(Mutation)**：在繁殖过程，新产生的染色体中的基因会以一定的概率出错，称为变异。变异发生的概率记为Pm 。例如：

变异前：

000001110000000010000

变异后：

000001110000100010000

**适应度函数 ( Fitness Function )**：用于评价某个染色体的适应度，用f(x)表示。有时需要区分染色体的适应度函数与问题的目标函数。例如：0-1背包问题的目标函数是所取得物品价值，但将物品价值作为染色体的适应度函数可能并不一定适合。适应度函数与目标函数是正相关的，可对目标函数作一些变形来得到适应度函数。

|  |
| --- |
| 基本遗传算法伪代码  /\*  \* Pc：交叉发生的概率  \* Pm：变异发生的概率  \* M：种群规模  \* G：终止进化的代数  \* Tf：进化产生的任何一个个体的适应度函数超过Tf，则可以终止进化过程  \*/  初始化Pm，Pc，M，G，Tf等参数。随机产生第一代种群Pop    do  {  　　计算种群Pop中每一个体的适应度F(i)。  　　初始化空种群newPop  　　do  　　{  　　　　根据适应度以比例选择算法从种群Pop中选出2个个体  　　　　if ( random ( 0 , 1 ) < Pc )  　　　　{  　　　　　　对2个个体按交叉概率Pc执行交叉操作  　　　　}  　　　　if ( random ( 0 , 1 ) < Pm )  　　　　{  　　　　　　对2个个体按变异概率Pm执行变异操作  　　　　}  将2个新个体加入种群newPop中  } until ( M个子代被创建 )  用newPop取代Pop  }until ( 任何染色体得分超过Tf， 或繁殖代数超过G ) |